A case study in applying discrete control synthesis to excavator operation

Hervé MARCHAND
VERTECS, IRISA / INRIA, 35042 RENNES, France
Herve.Marchand@irisa.fr www.irisa.fr/vertecs

Éric RUTTEN
BIP, INRIA Rhône-Alpes, 38330 MONTBONNOT, France
Eric.Rutten@inrialpes.fr www.inrialpes.fr/bip

Abstract— *Robotic and control systems are ever more complex to design, program, as well as to operate. Existing theoretical work and tool support in discrete control synthesis can be applied to improve task-level robot programming. This requires to determine patterns of tasks and objectives, which are at once domain-specific to robotics, and generic enough to cover a broad class of control systems. We illustrate such a framework by a case study concerning the interactive discrete control of tasks in an excavating system.*

I. MOTIVATION

Robotic systems, and more generally control systems, are getting more and more complex to design, to program and to operate. One aspect of this is that they feature numerous control tasks, corresponding to different functionalities, defined by different control laws. These tasks must be managed, in the sense that they have to be started, initialized, and exceptions observed and treated. In the case of complex missions, they also have to be sequenced or composed in parallel, and their interactions have to be controlled. Different approaches exist to robot architectures [6] offering support to the design of such controllers. Some of them rely on the use of discrete models (finite state automata, reactive systems and languages) for the specification, verification, and code generation. One of them is ORCCAD (Open Robot Controller CAD)[1] [3],which relies amongst others on the synchronous languages and tools [7].

Different techniques exist exploiting models based on finite state machines. Verification consists in determining, for a given property involving states or events, whether or not it is true for a given automaton, representing the possible behaviors of a program or system [2]. Discrete control synthesis, given the same as above, with the addition of a set of *controllable* events, consists of determining the constraints on the latter that make the resulting automaton satisfy the property, by inhibiting the transitions which would lead to its violation. The state of the art provides for general theoretical results and tool support, that can be applied to improve robot programming [5], [9]. Their practical application to robotics goes through specialization, so that it can be used by domain specialists, with no expertise in formal models. This requires patterns of tasks and objectives to be determined, which are at once domain-specific to robotics, and generic enough to cover a broad class of control systems.

This work is partially supported by the IST project TELEDIMOS .

[1] www.inrialpes.fr/iramr/pub/Orccad

This paper is proposes a generation process for task-level controllers, based on the use of available formal tools: SIGNAL and SIGALI. A user works with predefined task schemes, equipped with the necessary controllable events, as well as a number of properties of interest. The obtaining of the controller consists of a call to the automated control synthesis tool. Here we report on a case study concerning an excavation system, inspired by the TELEDIMOS project [12], and the partial automation of its operation by a human operator.

II. TASK-LEVEL ROBOTIC PROGRAMMING AND DISCRETE CONTROL SYNTHESIS

A. Robotic architectures

We consider robot control at task-level [6], where the continuous control part is encapsulated in a discrete, event-based control. Amongst related approaches, we take our inspiration in the structures used in ORCCAD [3], where the control algorithms are specified in terms of a data-flow network of numerical computations. These are encapsulated in a task with explicit phases or modes for initialization, nominal control, termination, and event handling for exceptions of different levels; They are modeled by a finite state automaton. Missions of a robot are called robot procedures, defined as combinations of such robot tasks, and can be specified using a domain-specific language like MAESTRO. This approach has been applied to manipulator arms, wheeled mobiles, an underwater ROV, automated cars [3].

In such an approach, the discrete behavior of the system is described by the automaton resulting from the composition of task automata according to the mission structure. Hence it is possible to use this automaton for purposes of analysis and design of the discrete aspect of applications. The advantage is that techniques for optimization, automated code generation, verification, are then applicable. ORCCAD is based on the synchronous models and technology [7], in particular the ESTEREL language and compiler, and verification tools FC2TOOL and/or CADP. Another applicable technique is then discrete control synthesis, which requires adaptations in the model, as well as an application methodology.

B. Discrete control synthesis

The basic models are discrete-event systems, and can be formulated as, e.g., formal languages, Petri nets, or finite state machine [4]. In the latter case, states cor-

respond to given configurations of the robot system, and/or of the activation status of tasks controlling it. Transitions correspond to the occurrence of, e.g., events, commands, thresholds. They are labels on the transitions, which can involve conditions on their fireability. The events labelling the transitions can be partitioned into those that can not be controlled (e.g. inputs received from sensors, failures) and those of which the value can be determined or constrained, typically by a discrete controller (typically the starting of some task). The former are called *uncontrollable*, and the latter *controllable*. In the sequel, we shall consider a transition systems, in which events can occur simultaneously. Hence a transition between two consecutive states can be labelled by a vector of events (some controllable, some others uncontrollable (See Fig. 2))

Properties and objectives. Such transitions systems can have properties related to the reachability of some subset of the state space, or to the existence of paths along which a certain sequence of events exists. They can concern invariants on the states themselves (i.e., the variables of which the valuation defines a state), or the paths that can be taken in the transition system from state to state, etc (See Section II-C.2). Synthesis will involve exploring how to constrain the controllable events in order to achieve an objective.

Synthesis algorithms and tools. The synthesis of a controller consists of automatically computing the *controller*: a relation that, given a state and uncontrollable events, gives the value of controlled events such that only transitions respecting the objectives can be taken (in other words: contradicting behaviors are inhibited), as illustrated in Figure 1 in the case of a deterministic controller. This produces a constrained model, i.e. model and controller together satisfy the property.In the framework depicted in Figure 1, the control strategy is the following: given a state and a set of uncontrollable events that occurs, the set of controllable events that may occur is given by the controller according to the restrictions on transitions computed during the synthesis phase.

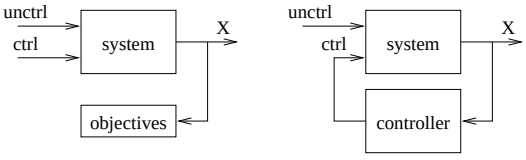

Fig. 1. Discrete control synthesis: from uncontrolled system (left) to closed-loop (right).

There can be several controllers satisfying the control objectives; actually, sometimes forbidding any move is a control which avoids the states not satisfying the property, but this is less than satisfactory w.r.t. the activity of the control system. The notion of maximally permissive controller captures that we have the controller which insures the properties satisfaction while keeping the greatest subset of behaviors of the original, uncontrolled, system.

Among the available methods and tools for discrete control synthesis, SIGALI is integrated with the SIGNAL environment for the design of real-time systems. It forms a complete toolset for undertaking experiments, as it provides for the quite rare combination of a high-level specification language, a complete compiler and code generator, and a formal computation tool performing discrete control synthesis [9].

C. Using discrete control synthesis in robotics

There are works on fundamental issues related to automated manufacturing regarding for example, resource management, boundedness, reversibility etc. Some studies have been experimented with on particular manufacturing systems [5]. Differently to these works, the goal here is to define a framework [11] for a pragmatic design assistance, based on existing theory and tools, not specific to an application, but generic enough for a wide range of robotic systems. In other words, the framework should be *domain* specific but not *application* specific. This generic aspect serves as a foundation for a design process where a tool support is integrated into the compilation from the high-level specification of tasks and missions (in terms of the application domain) to the actual, implemented discrete controller. Concretely, it consists of an assistance to model construction using task patterns equipped with control events, and a number of properties patterns that can be used as control objectives.

C.1 Discrete model of tasks and missions

In our discrete model of tasks and missions, we distinguish different discrete control states for each task, as shown in Fig. 2. Initially, each task is `Idle`. It goes from `Idle` to `Act` when there is a request (event `req`) and the controller accepts it (event `go`), i.e. the control constraints allow it. With the intention of "installing" controllability in the model, a `Wait` state has been incorporated to enable the recording of a request when the activity of another task prevents the controller from starting it. The controller may choose to make it active once the conditions are favorable. Termination of a task is signalled by the event `stop`. Under this model, only the event `go` is assumed to be *controllable*; the others are *uncontrollable*.

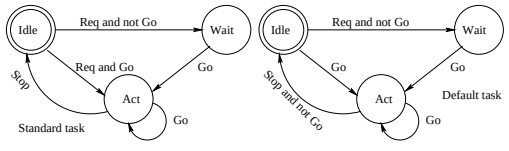

Fig. 2. The discrete model of the tasks.

Robots or control systems often require to be always under control, even for rest configurations, because of

gravity or other external forces. This motivates the introduction of a pattern for default tasks, shown in Fig. 2. It is similar to the standard task except that it is not necessary to have a request in order for a default task to become active. Also, when the event `stop` is triggered, the controller decides upon the termination of the task, by triggering `not go`.

Missions. In the present state of our framework, missions are obtained by the parallel composition of tasks, which builds a Cartesian product of automata. The synchronous composition of processes is defined exactly like that and therefore, using synchronous languages and compilation provides efficient tool support [7]. Given the looseness of this coupling, all tasks can be activated independently. For n tasks, the whole automaton is of size 3^n.

C.2 Objectives and synthesis for robot applications

The model includes amongst its configurations some that are undesirable, for example for reasons of resources to be shared (e.g., an actuator between different control laws), or criteria related to the functionality fulfilled by the tasks (e.g., incompatible side effects on the device or its environment). Amongst the paths described by sequences of transitions, there can also be undesirable ones, for example for reasons of necessary transitory modes between some tasks (e.g., between velocity-based and position-based movement control of a motor). In order to control these situations, we have to specify the properties on the states and events to be either achieved or avoided. Then, using them as synthesis objective, we can obtain the discrete controller, if it exists, which will constrain the behaviors in such a way that only those satisfying the properties will be allowed. Typical ones are for example:

Safety properties concerning state characteristics, like: the fact that for each actuator, one and only one control task must be active, or also that some tasks of one actuator must not be active together with some task of another actuator.

Reachability can be used to specify that the system can always be returned into an initial configuration;

Observers can be used to ensure that the system avoids a certain sequence of task activations. They are defined by an automaton, recognizing the sequence, with a terminal state. The global system is the parallel composition of the observer and the pre-existing system. It is submitted to an objective of safety keeping out of the terminal state.

Inhibiting two successive activations of the same task t_1 without carrying another one t_2 in between is an example of property. More precisely, we want to have, between the ending of t_1 and its next activation, at least a complete activation of t_2, from activation to end. Cases with simultaneity are acceptable. For this, an observer can be proposed as in Figure 3.

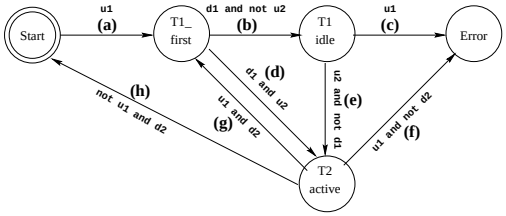

Fig. 3. Example of an observer process

This automaton, initially in state `Start`, observes changes in the value of state variables act_{t_1} and act_{t_2}, i.e., we have u_1 when act_{t_1} goes up from false to true (i.e., t_1 becomes active), and d_1 when act_{t_1} goes down from true to false (i.e., t_1 becomes inactive). Along the same lines, we have u_2 and d_2 for act_{t_2}. So we have the following transitions:

(a) a first occurrence of the starting of task t_1 causes a transition to state `T1_first`.

Upon the task deactivation d_1:

(b) in the absence of the activation of act_{t_2}, we go to state `T1_idle`. There,

(c) if task t_1 goes active again before an activation of t_2, the observer goes into the state denoting the error: `Error`.

(e) if act_{t_2} becomes active: u_2, and t_1 does not go active again, then we go to state `T2_active`.

(d) if simultaneously act_{t_2} becomes active: u_2, then we go to state `T2_active`. There,

(f) if t_1 goes active again before the end of t_2, the observer goes into the state denoting the error: `Error`.

(g) if t_1 goes active again simultaneously with t_2 ending, then we go to `T1_first`.

(h) if t_1 does not go active again when t_2 ends, and then the observation starts from the initial state again.

The desired property is that a sequence reaching `Error` is forbidden.

D. Discrete control synthesis with SIGNAL/SIGALI

We give here again just a few essentials, details being available elsewhere [9]. SIGNAL is a data-flow, equational language, with a graphical syntax in the form of block-diagrams. It is a high-level language to build models of discrete event systems. It provides programmers with hierarchical and compositional structures for the construction of large, complex systems. Some instructions used in the remainder of the paper are the composition of equations into systems : `|`, and functions, e.g., on Booleans, as in:

```
A := B or C | B := D and E
```

A delay operator can be seen as a register, and defines a discrete state. The behavior is such that in a given state the system can make a transition according to interface (input/output) and local events towards a new state. Its semantics is mathematically defined and underlies all

the analysis, verification and compilation techniques. It is implemented in a complete design environment featuring a graphical specification interface. Analysis and optimization tools can transform the original program automatically. Compilation and executable code generation are available for various execution platforms.

The discrete event dynamical aspects of SIGNAL have a model and theory based on polynomial dynamical equations systems. To every SIGNAL specification corresponds such a system, describing a transition system by initialization, evolution relations. Algebraic manipulations allow for the definition of operations checking satisfaction of properties like: invariance of a transition system w.r.t. a condition, reachability/attractivity of a set of states from another one. These notions can be used in control synthesis, where a transition system can modified by constraint on events declared controllable, *making* it satisfy a property [9]. A transition system can be submitted to a series of such operations, in a process of incremental synthesis. A tool is available, called SIGALI, which implements this with decision diagram techniques typical of model-checking. Some instructions use in the remainder are: `B_True` (resp. `B_False`) which designates the set of states where a predicate is true (resp. false), and `S_Security`, resp. `S_Reachable`, the synthesis operations for objectives of invariance, resp. reachability. The result of synthesis operations is a decision diagram, characterizing the constraints on controllable events necessary for the property to be satisfied (if possible). Specification, synthesis,

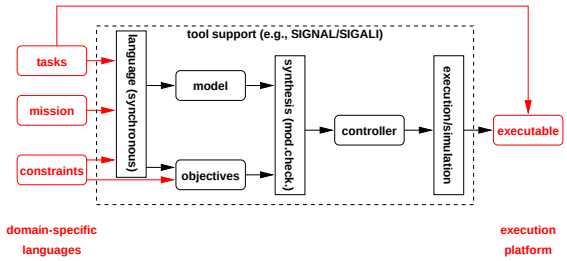

Fig. 4. The discrete control synthesis platform

and simulation in the SIGNAL/SIGALI environment goes as shown in Fig. 4 [9]. It involves the modeling of the system in all its possible behaviors and the specification of properties (desirable and undesirable, i.e., those *not* respecting properties) and objectives (invariance, reachability, attractivity). The properties and objectives can be expressed in SIGNAL, which eases their specification: they can be stated in terms of the variables and events used in the system model. The SIGNAL compiler is then used to produce a transition system given as input to SIGALI, upon which discrete control synthesis is performed automatically. The resulting controller is produced in a form that is recognized by a generic evaluator, which can be integrated with an application-specific graphical simulation environment. The SIGNAL compiler is used once again for the production of an interactive graphical simulator integrating model and controller. Modifying the specification and obtaining a new controller can be done automatically, by running the same operations, without having to re-examine the whole controller manually.

III. APPLICATION TO AN EXCAVATION SYSTEM

A. Model of the system

We consider a simplified model of an excavator system inspired by the TELEDIMOS project [12]. It provides us with an example which is manageable and illustrative, and at the same time showing some complexity. The system decomposes into sub-systems, according to the actuators: the articulated arm, the grip held at the end of the arm, the rotating cabin on which the arm is mounted, and the mobile base, carrying the whole, itself composed of two tracks. Each subsystem i.e., actuator, is equipped with a library of control tasks, corresponding to different functionalities of the device, and different ways of achieving them, according to different criteria. The complete excavator system is simply constituted by the composition of all its actuators, each with its control tasks, as shown in Fig. 5.

The grip is equipped with 2 tasks. The manipulation task `Manip` is a standard task; it is manually controlled i.e., the operator directly decides on the movements of the grip. At the lower level, the purpose of this task is to either open or close the grip. The task `Maint` is the default task; it maintains the grip at the current opening position.

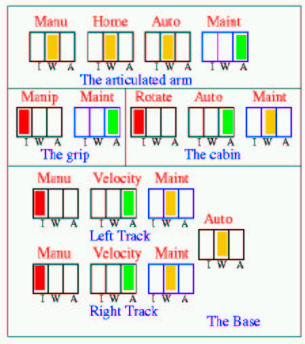

Fig. 5. The model of the excavator tasks.

The articulated arm is equipped with 4 tasks. The manipulation task `Manu` offers manual control to the operator e.g., with force-feedback, and possibly sharing of degrees of freedom with an automated control law in computer assisted teleoperation. The task `Home` brings the arm from whatever position it is in, towards a predefined resting position (e.g., folded in a position where a mechanical brake or blocking device can be put in place). The automated movement task `Auto` is defined

by a control law, for example, trajectory following, or sensor-based movement, with the sensor placed at the end of the arm. The task `Maint`, the default task, maintains the current position. It constitutes an actual control task because just cutting off power might result in the arm falling down due to gravity or moving arbitrarily due to strong wind or water current.

The rotating cabin is equipped with 3 tasks. The rotation control task `Rotate` offers manual control to the operator by means of handles or buttons, clockwise or counter clockwise. The automated movement task `Auto` determines the movement of the cabin automatically by a control law. A maintaining task `Maint` is the default task. It maintains the position of the cabin at the current angular position against any external torque.

The mobile base is composed of 2 sub-systems: the left and the right tracks. Each track is equipped with 3 independent tasks. The manual control task `Manu` gives control to the operator, similarly to the manual tasks of the grip or the arm. The velocity-based control task `Velocity` is such that velocity can be different for each track, resulting in the turning motion of the base. A maintaining task `Maint` is the default task; it maintains the current position of the base (e.g., on the slope of a hill), similarly to the maintaining tasks for the other actuators. There also exists the `Auto` task, which implements a control law controlling both the tracks together.

Each task is modeled in SIGNAL by an instance of the process encoding the appropriate task pattern (standard or default), as introduced above. Each subsystem is described by means of a composition of its tasks, sharing inputs and outputs. The composition of subsystems simply defines the whole system. Hence it describes, at that abstraction level, all possible dynamical behaviors of the excavation system.

B. Properties and objectives

Having each actuator always under control of one single task is characterized by an expression where at least one and only one `Act` signal is `true`. For example, in the case of the cabin, we define the Booleans:

```
| cabin_act := Act_auto_cabin
   or Act_rotate_cabin or Act_maint_cabin
| cabin_red := (Act_auto_cabin and Act_rotate_cabin)
   or (Act_auto_cabin and Act_maint_cabin)
   or (Act_rotate_cabin and Act_maint_cabin)
```

The predicate that is to be ensured to be `true` is then,

```
Ctrl_cabin := cabin_act and (not cabin_red)
```

Finally, the SIGALI macro to compute the controller ensuring the trueness of the above predicate (and thus the *invariance* of the corresponding set of states) is:

```
SIGALI(S_Security(B_True(Ctrl_cabin))
```

The control of the grip, the tracks and the cabin is similar. Other properties concern the interactions between different subsystems. Safety concerns can command to exclude manually controlled movements of an actuator when the others upon which it is mounted are in movement. This decomposes into:

1. no cabin movement (except automated) while driving (i.e. when the base is moving, i.e. when it is in any other task than `Maint`):

```
| Driving := not (Act_maint_track_left
                  and Act_maint_track_right)
| cabin_base := Act_rotate_cabin and Driving
| SIGALI(S_Security(B_False(cabin_base)))
```

2. No arm movement (except automated) while driving or rotating:

```
| Rotating := not Act_maint_cabin
| arm_cabin_base := Act_manu_arm
        and (Driving or Rotating)
| SIGALI(S_Security(B_False(arm_cabin_base)))
```

3. or similarly: No manual manipulation of the grip when arm movement, driving or rotating:

The initial state of the system is that in which all the default tasks of the subsystems are active and all other control tasks are idle. An interesting control objective is to ensure the reachability of a state where all and only the default tasks are active, seen as a safe resting state. Note that this state is not exactly the initial state since some of the other tasks may be in the `Wait` state, but it can characterize a restart state. Specifying the reachability of this state is as follows:

```
| config_init := Act_maint_grip and Act_maint_arm
     and Act_maint_cabin and Act_maint_track_left
     and Act_maint_track_right
| SIGALI(S_Reachable(B_True(config_init)))
```

We want to exclude two successive manipulations of the grip without any cabin or base movement in between. We make use of an *observer* as in Fig. 3. Instantiating this observer with t_1 as `Act_manip_grip` and t_2 as the conjunction of `Act_maint_cabin`, `Act_maint_track_left` and `Act_maint_track_right`, we obtain the desired automaton, in which the final state `Error` is reachable only after the system goes through the specified task sequence to be avoided. Thus we want the invariance of the set of states where `Error` is `false`, i.e., for synthesis: `SIGALI(S_Security(B_False(Error)))`

C. Interactive simulation

At every phase in the above construction of the model and control, it is possible to obtain a simulator of the behaviors of the controlled system. This way, a user can observe how the behaviors change when adding one objective, and verify whether the constraints added correspond to the problem to be solved. A simulation consists of iterating, step by step, the following three operations:

1. simulating the environment is done through the uncontrollable inputs panel (see Fig. 6 (left)), where one can enter the requests from the operator, and the events signalling termination of tasks.
2. choosing among correct controls is done through the controllable events panel (Fig. 6 (right)). Values ruled

out by constraints are represented by non-selectable buttons. There can be possibly several possible values, if the constraints do not completely determine the control from the input. Then, the control of the system is directed in a shared way by partly requests from the user, partly control by the automatism. This is reminiscent of the concept of teleoperation, where movements of an actuator are directed by a human operator, with a sharing of some degrees of freedom with automated control laws. As such, this mode of interactive control can be called discrete teleoperation.

Fig. 6. Interactive Panels.

3. the dynamical evolution is observed with the task states display shown in Fig. 5. Changes in behaviors obtained after adding a control objective can show in, e.g., observing that a requested task goes into wait state when another, incompatible one is active, until termination.

IV. Conclusion and perspectives

This paper illustrates a framework for using discrete controller synthesis in safe control systems programming, at the level of a set of control tasks. We have proposed a set of task and properties patterns, which correspond to concepts used by application designers. These patterns can be used for the construction of application models, and the application of discrete control synthesis computation, without requiring deep knowledge of the underlying formal models and theory. In that sense, it proposes a user-friendly, automated use of this formal technique. The framework is systematic enough to design and develop an automated process on these bases. It could be integrated with a programming environment like ORCCAD. We have built up a practical experiment, using the SIGNAL/SIGALI environment. The obtained controllers are simulated in an interactive way, which can be interpreted as a form of teleoperation, where the control is shared between a human operator and an automatism, in a discrete space. The case study, inspired by a excavation system [12], was treated to confront the framework with an example of some size, while keeping manageable and observable. It illustrates the construction of the model of behaviors, the specification of properties and objectives, and the use of the obtained controller in a interactive simulation environment.

Perspectives and further developments of the approach concern a mission language which would allow for the specification of sequence patterns for tasks, for partial specification of missions less “loose” than just parallel composition, but still keeping some freedom, within which discrete control synthesis would care for lower-level management of the avoidance of conflicts, following properties like the ones described in this paper. An execution framework for the synthesized controller would be an interesting progress from this simulation framework. Other synthesis techniques could bring significant enrichment in the task-level modeling of robot systems, e.g., multi-mode tasks, with different implementations of a control law distinguished by cost (e.g., computational, energetic) and quality (e.g., accurateness) [8]; the use of optimal synthesis can make use of this information‘[10]. Also of interest are the taking into account of fault tolerance aspects, and hybrid systems [1].

References

[1] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings of the IEEE*, 88, 2000.

[2] .B Bérard et al. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[3] J-J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro. The ORCCAD architecture. *Int. J. of Robotics Research*, 17(4), 1998.

[4] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[5] F. Charbonnier, H. Alla, and R. David. The supervised control of discrete event dynamic systems. *IEEE Trans. on Control Systems Technology*, 7(2):175–187, March 1999.

[6] E. Coste-Manière and B. Espiau. Special issue on integrated architecture for robot control and programming. *Int. J. of Robotics Research*, 17(4), April 1998.

[7] N. Halbwachs. *Synchronous programming of reactive systems*. Kluwer, 1993.

[8] F. Maraninchi, Y. Rémond, and E. Rutten. Effective programming language support for discrete-continuous mode-switching control systems. In *Proc. of the 40th IEEE Conf. on Decision and Control, CDC’01*, Orlando, Florida, dec, 2001.

[9] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic. Synthesis of discrete-event controllers based on the signal environment. *Discrete Event Dynamical System: Theory and Applications*, 10, October 2000.

[10] H. Marchand and E. Rutten. Managing multi-mode tasks with time cost and quality levels using optimal discrete control synthesis. In *14th Euromicro Conference on Real-Time Systems (ECRTS’02)*, June 2002.

[11] Éric Rutten and Hervé Marchand. Task level programming for control systems using discrete control synthesis. Research Report 4389, INRIA, February 2002. http://www.inria.fr/rrrt/rr-4389.html.

[12] D. Simon, M. Personnaz, and R. Horaud. TELEDIMOS: telepresence simulation platform for civil work machines: real-time simulation and 3d vision reconstruction. In *Proc. IARP Workshop on Advances in Robotics for Mining and Underground Applications, Brisbane, Australia, Oct.*, 2000.